Blurred Encoding for Trajectory Representation Learning

Silin Zhou
University of Electronic Science and Technology of China
Chengdu, China
zhousilinxy@gmail.com

Yao Chen†
National University of Singapore
Singapore
yaochen@nus.edu.sg

Shuo Shang†
University of Electronic Science and Technology of China
Chengdu, China
jedi.shang@gmail.com

Lisi Chen
University of Electronic Science and Technology of China
Chengdu, China
lchen012@e.ntu.edu.sg

Bingsheng He
National University of Singapore
Singapore
hebs@comp.nus.edu.sg

Ryosuke Shibasaki
LocationMind Inc.
Tokyo, Japan
shiba@locationmind.com

Abstract

Trajectory representation learning (TRL) maps trajectories to vector embeddings and facilitates tasks such as trajectory classification and similarity search. State-of-the-art (SOTA) TRL methods transform raw GPS trajectories to grid or road trajectories to capture high-level travel semantics, i.e., regions and roads. However, they lose fine-grained spatial-temporal details as multiple GPS points are grouped into a single grid cell or road segment. To tackle this problem, we propose the **BLU**rred **E**ncoding method, dubbed **BLUE**, which gradually reduces the precision of GPS coordinates to create *hierarchical patches* with *multiple levels*. The low-level patches are small and preserve fine-grained spatial-temporal details, while the high-level patches are large and capture overall travel patterns. To complement different patch levels with each other, our BLUE is an encoder-decoder model with a pyramid structure. At each patch level, a Transformer is used to learn the trajectory embedding at the current level, while *pooling* prepares inputs for the *higher level* in the encoder, and *up-resolution* provides guidance for the *lower level* in the decoder. BLUE is trained using the trajectory reconstruction task with the MSE loss. We compare BLUE with 8 SOTA TRL methods for 3 downstream tasks, the results show that BLUE consistently achieves higher accuracy than all baselines, outperforming the best-performing baselines by an average of 30.90%. Our code is available at https://github.com/slzhou-xy/BLUE.

CCS Concepts

• **Information systems** → **Spatial-temporal systems**; • **Computing methodologies** → **Artificial intelligence**.

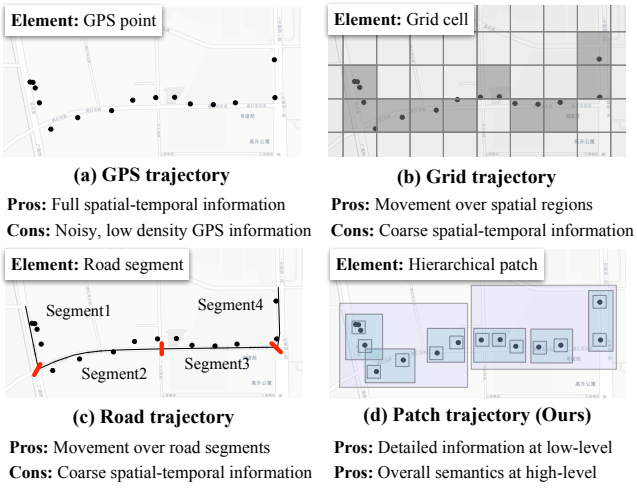


Figure 1: An illustration of different trajectory expressions.

Keywords

Trajectory representation learning; Hierarchical patches; Multiple levels; Pyramid structure

ACM Reference Format:
Silin Zhou, Yao Chen, Shuo Shang, Lisi Chen, Bingsheng He, and Ryosuke Shibasaki. 2025. Blurred Encoding for Trajectory Representation Learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3736861

1 Introduction

With the popularization of GPS-enabled devices, human and vehicle movement trajectories can be easily collected. Analyzing these trajectories is crucial for many urban tasks such as traffic management [35], business services [48], and regional planning [24], which contribute to smarter cities [6]. As a fundamental task for trajectory analysis, *trajectory representation learning* (TRL) generates a vector for each trajectory while preserving the spatial-temporal travel semantics of the trajectory [5, 27]. By transforming variable-length trajectories into fixed-length vectors, TRL supports many downstream tasks, such as trajectory classification [25, 44], travel time

†Shuo Shang and Yao Chen are corresponding authors.

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3736861

estimation [16, 28], trajectory similarity computation [17, 47], etc. For instance, trajectory similarity computation originally requires quadratic time w.r.t. trajectory length but becomes linear-time Euclidean distance computation with TRL [19].

Existing works and their limitations. The SOTA TRL methods use either grid trajectories [3, 23, 43] or road trajectories [20, 26, 31] as inputs to tackle the irregularities and noises of GPS trajectories. Figure 1(a) shows a raw GPS trajectory, which is a time-ordered sequence of GPS points. However, these points are spatially and temporally irregular due to variations in sampling intervals and positioning noise, and each GPS point carries limited spatial-temporal information and travel semantics [5]. Thus, directly modeling GPS trajectories yields low accuracy. As shown in Figure 1(b), a grid trajectory is a sequence of grid cells, where each cell corresponds to a region of the city and groups multiple GPS points. By grouping GPS points, grid trajectories reduce the influence of jitters and noises and can capture the high-level properties of regions (e.g., living or commercial areas) [5]. Figure1(c) illustrates a road trajectory, which consists of road segments obtained by applying map-matching [4] on the raw GPS trajectories. Road trajectories represent object movement along the road network, with each segment corresponding to a street and grouping multiple GPS points.

Although grid and road trajectories benefit TRL by capturing high-level travel semantics, i.e., over regions or roads, they also suffer from three crucial limitations. ❶ *Limited resolution.* By grouping multiple GPS points into a grid cell or road segment, they capture high-level local information but lose fine-grained spatial-temporal details inherent to individual GPS points, which is essential for preserving basic spatial-temporal properties. In particular, a grid cell represents a geographical region and thus cannot assign precise timestamps to it. Similarly, a road segment usually only retains the average time of its GPS points [5]. The limited spatial-temporal resolution hinders tasks that require fine-grained information (e.g., travel time estimation). ❷ *Poor generalization.* The grid cells and road networks differ for cities, and thus TRL models that use grid or road trajectories cannot generalize across different cities. However, a good generalization is favorable for reducing model training costs and handling cases when data is scarce. ❸ *Poor robustness.* Grid-based TRL methods need to tune the size of grid cells for good accuracy [19, 38]. The quality of the road trajectories depends on the map-matching algorithm and the precision of the road graph [30, 40], and thus the accuracy of road-based TRL methods is also affected. Given the shortcomings of grid and road trajectory expressions, we pose the following research question:

> *Can we design a trajectory expression that attains both low-level spatial-temporal details and high-level travel semantics for accurate, general, and robust TRL?*

Our solution BLUE. We propose the ***blu**rred **e**ncodin**g*** method, dubbed *BLUE*, to transform each GPS trajectory into a *patch trajectory*. As shown in Figure 1(d), our patches have a *hierarchy*, with each low-level patch covering a small area and a high-level patch encompassing multiple low-level patches. Moreover, each patch may contain a different number of GPS points. In particular, we drop the least significant decimals of the GPS coordinates and group the points that share a common prefix into one patch. By *progressively* dropping more decimals for the patches in higher levels, we form the patch hierarchy. Using the low-level patches, our patch trajectory preserves the fine-grained details of the GPS trajectory, and using the high-level patches, the patch trajectory can capture high-level regions and travel semantics. Moreover, since high-level patches encompass low-level patches, there are opportunities for two adjacent patch levels to complement each other to improve the TRL model. Finally, the patch trajectory does not rely on grid size or map matching and thus is more general and robust.

Based on the patch trajectory, we design the BLUE model, which has an encoder and decoder with a pyramid structure. In particular, the encoder progressively compresses and transforms low-level GPS trajectories into high-level patch trajectories with multiple model levels. Each model level involves two components, i.e., a *Transformer* to capture global trajectory information at the current level, and a *patch pooling* to capture local semantics of the patches and generate higher-level patch trajectory. Conversely, the decoder progressively reconstructs low-level GPS trajectories from high-level patch trajectories. Each level of the decoder encompasses a *up-resolution* to recover the current-level trajectory from a higher level, and a *Transformer* to refine the restored trajectory for improved accuracy. To handle variable patch lengths in patch pooling and different trajectory lengths during up-resolution, we design an attention-based method for patch pooling and use cross-attention for up-resolution. To train our BLUE, we employ the mean squared error (MSE) loss for reconstructing the raw GPS trajectories, which is more efficient than the cross-entropy loss (involving a large number of classes with the Softmax [33]) used in grid-based and road-based TRL methods.

To evaluate BLUE, we compare it with 8 SOTA TRL methods for 3 popular tasks, i.e., travel time estimation [31], most similar trajectory search [3], and trajectory classification [20]. The results show that BLUE consistently outperforms all baselines across the tasks and datasets, with an average accuracy improvement of 71.57%, 14.48%, and 1.82% for travel time estimation, most similar trajectory search, and trajectory classification, respectively. Ablation studies confirm the effectiveness of BLUE's designs. Generalization tests show that BLUE performs much better than all baselines when it is trained on trajectories from one city but applied to another city, i.e., in the case of transferability.

To summarize, we make the following contributions:

- We observe that SOTA TRL methods use either grid or road trajectories, which have problems including limited resolution, poor generalization, and poor robustness.
- To tackle the problems of grid and road trajectories, we propose blurred encoding to generate patch trajectories, which have hierarchical levels by progressively dropping GPS decimals and thus capture both low-level details and high-level semantics.
- Based on patch trajectories, we design a TRL model named BLUE, which has an encoder and decoder with a pyramid structure, to augment adjacent levels of patch trajectories with each other.

2 Related Work

Early TRL studies [29, 45] use the raw GPS trajectories. For example, Traj2vec [45] generates movement features from GPS points with an RNN-based Seq2Seq [36] model. CSTRM [29] introduces point-level and trajectory-level differences to learn the trajectory

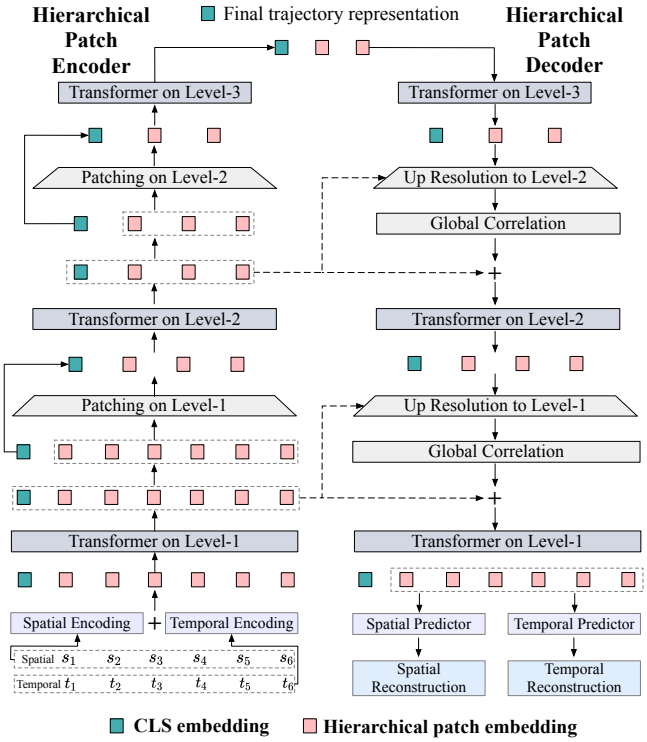


Figure 2: An overview of the BLUE model.

representation. However, these methods are usually designed for a specific task, i.e., Traj2vec for clustering and CSTRM for trajectory similarity computation, thus lacking generality. More importantly, the amount of information carried by a single GPS point is limited, and thus directly modeling it does not yield high accuracy [5]. To solve these problems, SOTA TRL methods focus on using the grid or the road trajectories derived from the GPS trajectories.

TRL with grid trajectories. To simplify and enhance GPS trajectories in free space, some methods [3, 12, 23, 43] use the grid cells to represent regions of a city map, and further group the multiple GPS points in the same region. Therefore, the GPS trajectory becomes the grid cell sequence, which introduces the region information. For example, t2vec [23] uses an encoder-decoder model [36] to learn robust grid trajectory representations from low-quality data by a spatial proximity aware loss function and a cell pre-training algorithm. E^{2}DTC [12] proposes self-training to capture hidden spatial dependencies and learn grid trajectory representations. TrajCL [3] uses contrastive learning [15] with various trajectory-specific augmentations to learn grid trajectory representations by introducing spatial and structural perspectives from GPS information. Although grouping points into a grid cell can reduce the GPS noise [5] and introduce region information, it changes the basic unit of the GPS trajectory and thus blurs the spatial-temporal information.

TRL with road trajectories. Another way to reduce GPS noise and enhance the GPS trajectory is to use the map matching [40] algorithm to group multiple GPS points to a road segment to get the road trajectory. The road trajectory is continuous in the road network and can reflect the transition of object motion. Moreover, the road segment of the road network introduces the city topology and street information [27]. Therefore, most TRL studies [7, 13, 20, 32, 41, 42] are based on the road trajectory. For example, Trembr [13] uses an RNN-based seq2seq model [36] to learn spatial-temporal information of the road trajectory. PIM [41] introduces the graph embedding by node2vec [14] to learn road segment embeddings and designs the mutual information maximization to learn the road trajectory representation. JCLRNT [32] utilizes three types of contrastive loss, i.e., road-road, road-trajectory, and trajectory-trajectory, replacing detours and removing alternatives to learn the road trajectory representations. START [20] is a BERT-based [9] model and incorporates travel semantics and temporal regularities with various data augmentations of the road trajectory. Although the road trajectory introduces additional information, it also changes the basic unit and has the problem of limited spatial-temporal information.

Recently, some multi-modal methods [26, 31] have been proposed to conduct TRL from multiple views. For example, JGRM [31] learns trajectory representations by aligning the trajectory views of free space and road network jointly. MMTEC [26] proposes an attention-based discrete encoder and a NeuralCDE-based [21] continuous encoder to extract the travel behavior and continuous spatial-temporal correlations of the trajectories. However, they are still limited by using road trajectories. Different from existing TRL methods, our model provides a new trajectory expression, i.e., hierarchical patches, to capture the high-level semantics of the trajectories without losing the low-level details.

3 Problem Definition

Definition1 (GPS Trajectory). A GPS trajectory T is a time-ordered sequence of GPS points collected at a fixed or an unfixed sampling time interval by a GPS-enabled device. Each GPS point $g_i \in T$ is (x_i, y_i, t_i), where x_i, y_i, and t_i denote longitude, latitude, and timestamp, respectively.

Definition2 (Trajectory Representation Learning). Given a set of GPS trajectories, trajectory representation learning (TRL) aims to learn a d-dimensional vector representation for each trajectory in the set. We expect learned representations to achieve high accuracy for downstream tasks, e.g., travel time estimation, trajectory classification, most similar trajectory search, etc.

4 The BLUE Model

Figure 2 shows the overall structure of BLUE, which is an encoder-decoder model with a pyramid structure, consisting of four key modules: 1) Spatial-temporal Embedding: Transforms spatial-temporal GPS points into hidden embeddings. 2) Patch Encoder: Learns hierarchical spatial-temporal patch embeddings by progressively reducing GPS precision to create patches from high to low resolution. 3) Patch Decoder: Restores high-resolution trajectories from compressed low-resolution patches using cross-attention for length-insensitive restoration. 4) Spatial-temporal Reconstruction: Trains the model by reconstructing spatial-temporal GPS trajectories with MSE loss. For simplicity, multi-head attention and the [CLS] token are omitted in the Transformer formulas.

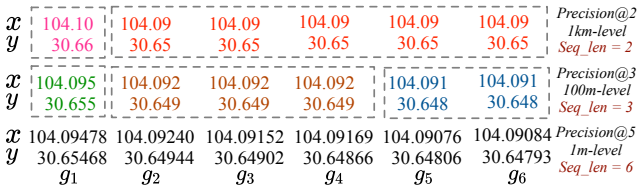


Figure 3: An illustration of blurred encoding by progressively rounding the decimals of the GPS points. The precision@5 is the raw GPS trajectory (i.e., level 1), while precision@3 and precision@2 are the patch trajectories at level 2 and level 3.

4.1 Blurred Encoding

Before introducing the backbone network, we present blurred encoding. Inspired by patches [10, 34], we propose a trajectory-specific patch method based on different decimal precisions of GPS to progressively drop the least significant decimals of a GPS coordinate to decide the hierarchical patches. It generates patch trajectories to learn high-level travel semantics and preserves the GPS trajectories to learn detailed low-level spatial-temporal semantics.

Figure 3 illustrates the details of blurred encoding. Specifically, we consider three precision levels: precision@5 (level-1) with 1m resolution, representing the raw GPS trajectory; precision@3 (level-2) with 100m resolution, and precision@2 (level-3) with 1km resolution, denoting the patch trajectory. Precision@4 (10m resolution) is omitted as it shows no significant difference from precision@5. To transition from precision@5 to precision@3, we use a rounding method that groups GPS points with similar spatial semantics at precision@5 into the same patch at precision@3. This process transforms a GPS trajectory into a patch trajectory. For example, g_2, g_3, and g_4 at precision@5 are grouped into the same patch at precision@3. This grouping process is repeated to generate new patch trajectories at precision@2, capturing higher-level travel semantics. Notably, the input remains at precision@5, as lower precisions are not used directly but rather inferred through these mappings[1].

This method leverages the semantics carried by GPS points to group low-level points into patches by blurring the GPS resolution. As a result, it offers several advantages: ❶ *Adaptive patch length.* The patch length adjusts dynamically based on GPS semantics and density by precision, making it flexible for trajectories of varying lengths. For example, a trajectory with many clustered GPS points will have a longer patch length, while a trajectory with fewer, widely spaced points may have a patch length of 1. ❷ *Semantic consistency.* GPS points within a patch share consistent low-level semantics, representing meaningful local properties. This reduces the spatial-temporal inconsistencies in the raw GPS trajectory and avoids truncating movement semantics. ❸ *Hierarchical representation.* Patch trajectories at different levels reveal unique information, reflecting hierarchical semantics. For instance, level-1, i.e., precision@5, retains basic spatial-temporal details, level-2, i.e., precision@3, captures local movement behaviors like turns, and level-3, i.e., precision@2, provides insights into long-range overall patterns. Moreover, the patch trajectory length at high-level is also greatly reduced, thus improving efficiency.

Compared to patches in computer vision (CV) and time series forecasting (TSF), our blurred encoding solves two issues in trajectories. ❶ *Variable length.* In CV, batches of images with the same resolution of 512×512 can be converted into patch sequences of length 64 using a fixed patch size of 64×64 [10]. Similarly, in TSF, batches of sequences with a fixed length of 128 can be transformed into patch sequences of length 16 using a patch size of 8 [34]. However, trajectory batches vary significantly in length, ranging from just a few points to several hundred, making it impractical to define a fixed patch size for them. ❷ *Local semantic.* In CV, a patch captures local spatial semantics, while in TSF, a fixed patch length (e.g., 12 for a 5-minute sampling rate) represents one hour of temporal semantics. However, for trajectories, a fixed patch length struggles to capture meaningful movement behavior, often segmenting dense clusters or continuous points, disrupting movement semantics.

[1]To avoid noun confusion, we use the terms *low-level* and *high-level* to refer to high-resolution (i.e., high-precision) and low-resolution (i.e., low-precision), respectively, in the following sections.

4.2 Spatial-temporal Embedding

The spatial-temporal embedding layer aims to convert coordinates and timestamps of the GPS trajectory, i.e., level-1, into spatial-temporal hidden embeddings. We design two encoding modules to achieve this: spatial encoding and temporal encoding.

Spatial Encoding. The raw GPS point g_i only has limited spatial information, i.e., the coordinate. To enrich spatial information, we introduce additional spatial relations within a trajectory. In particular, for a GPS point g_i, we compute its *forward distance* and *forward azimuth* to the next GPS point g_{i+1}, and its *backward distance* and *backward azimuth* to the previous GPS point g_{i-1} of a trajectory. If g_i is the first point of a trajectory, the backward distance and backward azimuth are zero, and if g_i is the last point of a trajectory, the forward distance and forward azimuth are zero. Therefore, combining the min-max normalized latitude and longitude information, we can obtain a spatial context vector $s_i \in \mathbb{R}^6$ for g_i in a trajectory. Then we use a linear layer as the spatial encoding layer to transform s_i to the spatial embedding as follows:

$$\mathbf{e}_i^s = \text{Linear}(s_i) \in \mathbb{R}^d, \tag{1}$$

Temporal Encoding. The raw GPS time information of a GPS point g_i is a timestamp, including the year, month, day, hour, minute, and second. However, the gap between these values is large. For example, the period of a year is usually 365, and the period of a second is 60, resulting in inconsistent input data ranges. To solve this issue, we use *day-of-year*, *day-of-month*, *day-of-week*, *hour-of-day*, *minute-of-hour*, and *second-of-minute*, to transform each value to the same range $[0, 1]$, we further subtract 0.5 to keep the range between $[-0.5, 0.5]$ of each value. Therefore, we obtain a temporal context vector $t_i \in \mathbb{R}^6$ of a GPS point g_i. Then we encode temporal information as follows:

$$\mathbf{e}_i^t = \mathbf{W}_1 t_i \parallel \sin(\mathbf{W}_2 t_i), \tag{2}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{\frac{d}{2} \times 6}$ are used to learn temporal context information, $\sin(\cdot)$ is further used to learn time periodicity information, $\parallel$ is the concatenation operation, and $\mathbf{e}_i^t \in \mathbb{R}^d$. Therefore, the final input embedding is $\mathbf{e}_i = \mathbf{e}_i^s + \mathbf{e}_i^t$ for a GPS point g_i.

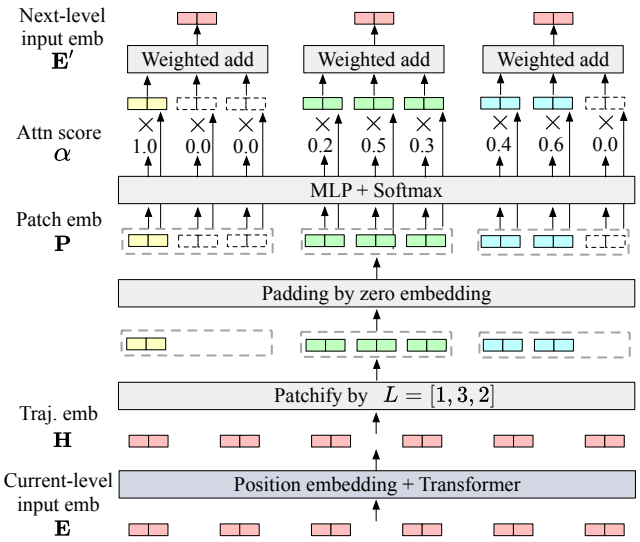


Figure 4: The patch encoder of BLUE from level-1 to level-2, the embedding dimension is 2, and the [CLS] token is omitted for simplicity. The encoder from level-2 to level-3 is similar.

4.3 Pyramid Encoder-decoder Structure

Our backbone network is an encoder-decoder model with a pyramid structure. The encoder is used to learn the trajectory representation from low-level to high-level by compressing the trajectory with blurred encoding. The decoder aims to restore the low-level trajectory from the compressed high-level trajectory.

Patch Encoder. The patch encoder alternates between the Transformer and blurred encoding. Figure 4 shows the pipeline of the encoder. The Transformer is used to capture global sequence information at the current-level. Blurred encoding by patching and pooling is to compress the trajectory embedding and generate the patch embedding with local semantics, then serve as the input for the next-level. In particular, we first add position encoding [37], then use the Transformer on GPS trajectory embedding $\mathbf{e}_i$ to learn basic spatial-temporal information at level-1 as follows:

$$\begin{aligned} \mathbf{E} &= [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_{|T|}], \\ \mathbf{H} &= \text{TransformerEncoder}(\mathbf{E}), \end{aligned} \tag{3}$$

where $\mathbf{H} \in \mathbb{R}^{|T| \times d}$ is trajectory embedding with basic spatial-temporal and global information at level-1. Then we blur it by patching $\mathbf{H}$ to the patch trajectory embedding as follows:

$$\begin{aligned} \mathbf{H} &= [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{|T|}], \\ L &= [l_1, l_2, ..., l_{|T_n|}], \\ \mathbf{P} &= \text{Padding}(\text{Patchify}(\mathbf{H}, L)), \end{aligned} \tag{4}$$

where $l_i \in L$ is a scalar that denotes a patch length for $\mathbf{H}$ and $\sum_{i=1}^{|T_n|} l_i = |T|$, which can be obtained using blurred encoding when loading data. $|T_n|$ is the length of the patch trajectory of the next-level, i.e., level-2. Patchify$(\cdot,\cdot)$ groups GPS point embeddings into a patch by using L. Padding$(\cdot)$ is used to keep the variable patch length the same. Therefore, $\mathbf{P} \in \mathbb{R}^{|T_n| \times M \times d}$, where M denotes the maximum patch length. To better explain Equation 4, we use a simple example, i.e., from level-1 to level-2, to illustrate it:

Example 1: The GPS trajectory $T = [g_1, g_2, ..., g_6]$ of level-1 is first learned to trajectory embedding $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_6]$ by using Equations (1)-(3). $L = [1, 3, 2]$ records the patch length from level-1 to level-2, where $M = 3$ and $\text{Sum}(L) = |T|$ (i.e., $\text{Sum}(L) = 6$). Then we use Patchify$(\mathbf{H}, L)$ to get patch embedding sequence $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$, where $\mathbf{p}_1 = [\mathbf{h}_1]$, $\mathbf{p}_2 = [\mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4]$, and $\mathbf{p}_3 = [\mathbf{h}_5, \mathbf{h}_6]$. We use padding embedding $\mathbf{h}^p$ to make each patch in $\mathbf{P}$ as the same length. Therefore, the final patch embeddings are $\mathbf{p}_1 = [\mathbf{h}_1, \mathbf{h}^p, \mathbf{h}^p]$, $\mathbf{p}_2 = [\mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4]$, and $\mathbf{p}_3 = [\mathbf{h}_5, \mathbf{h}_6, \mathbf{h}^p]$, i.e. $\mathbf{P} \in \mathbb{R}^{3 \times 3 \times d}$.

To generate trajectory embeddings at the next-level by patch embedding $\mathbf{P}$, we propose an attention-based pooling method to eliminate the effects of padding embedding and provide more effective pooling results. Specifically, the attention score for an embedding $\mathbf{p}_{ij} \in \mathbf{p}_i$ in a patch $\mathbf{p}_i \in \mathbf{P}$ is computed as follows:

$$\alpha_{ij} = \text{MLP}(\mathbf{p}_{ij}), \tag{5}$$

where the MLP consists of a linear layer, a layer normalization [2], a relu activation [1], and another linear layer, executed sequentially. In Example 1, $\mathbf{p}_{11} \in \mathbf{p}_1$ is $\mathbf{h}_1$. If α_{ij} is computed by padding embedding $\mathbf{h}^p$, we set α_{ij} as `float(-inf)`. Then we softmax-normalize attention-score α_{ij} in a patch as follows:

$$\alpha'_{ij} = \frac{e^{\alpha_{ij}}}{\sum_{k=1}^{M} e^{\alpha_{ik}}}, \tag{6}$$

If α_{ij} = `float(-inf)`, then $\alpha'_{ij} = 0$. The next-level input embedding can be obtained by using a weighted addition as follows:

$$\begin{aligned} \mathbf{e}'_i &= \sum_{j=1}^{M} \alpha'_{ij} \cdot \mathbf{p}_{ij}, \\ \mathbf{E}' &= [\mathbf{e}'_1, \mathbf{e}'_2, ..., \mathbf{e}'_{|T_n|}], |T_n| < |T|. \end{aligned} \tag{7}$$

$\mathbf{E}'$ is the patch trajectory embedding of level-2. *Noted that [CLS] token is ignored in patching of Equation(4)-(7), after patching, [CLS] token is prepended to patch trajectory embedding.*

We repeat Equations (3)-(7) to derive the patch trajectory embedding $\mathbf{H}'$ in Transformer at level-2 with global correlation and new patch trajectory embedding $\mathbf{E}''$ at level-3. Then Equation (3) is executed once again to learn the final compressed patch trajectory representation $\mathbf{H}''$ at level-3 to capture high-level travel semantics and global information. *The [CLS] token of $\mathbf{H}''$ is considered as the final trajectory representation and used for downstream tasks.* Noted that the sequence length is reduced, so positional encoding is still applied before each use of the Transformer.

Patch Decoder. The patch decoder is architecturally symmetric to the patch encoder, alternating between the up-resolution operation and the Transformer. The up-resolution operation is designed to restore the next low-level from the current high-level. The Transformer is employed to capture global correlations in the restored sequence, enhancing reconstruction capabilities. Notably, we first apply a Transformer to derive the new trajectory embedding $\widehat{\mathbf{H}}''$ from the final trajectory representation $\mathbf{H}''$, ensuring: 1) Symmetry in the architecture. 2) A deeper patch decoder structure.

Next, we employ an up-resolution operation to restore the next low-level from the current high-level, serving as the inverse process of patching in the patch encoder. In CV, up-resolution can be

achieved using techniques like up-pooling [46], transposed convolution [11], etc. However, these approaches operate on fixed input and output sizes, such as transitioning from a low-resolution of 64×64 to a high-resolution of 512×512. In trajectory data, the actual lengths of trajectories are variable. For example, a batch of trajectories has a shape of (b, l), where b is the batch size (number of trajectories) and l is the maximum trajectory length. In this batch, most trajectories are shorter than l, requiring padding to equalize their lengths. Due to this variability in length, up-resolution used in CV cannot be directly applied to trajectories. To clarify this, consider a simple example as follows:

Example 2: The shape of compressed trajectories at high-level is $(3, 6)$, where 3 represents the batch size and 6 is the maximum length in this batch. However, the actual length of compressed trajectories is $[4, 6, 3]$. Similarly, the shape of expected restored trajectories at low-level is $(3, 15)$, where 15 is the maximum length of restored trajectories in this batch. The actual length of expected restored trajectories is $[15, 12, 10]$. No direct length correspondence can be established between high-level and low-level trajectories.

To address the above issue, we introduce an up-resolution operation based on the cross-attention. Specifically, the compressed trajectory embeddings are used as input, while the trajectory embeddings from the patch encoder at the corresponding resolution of the expected restored trajectories are leveraged to assist in the restoration process. The operation of restoration from level-3 to level-2 is formulated as follows:

$$\widehat{\mathbf{E}}' = \text{CrossAttn}(\text{query} = \mathbf{H}', \text{key} = \widehat{\mathbf{H}}'', \text{value} = \widehat{\mathbf{H}}''), \tag{8}$$

where $\mathbf{H}'$ is the query, which comes from the output of the Transformer at level-2 of the patch encoder. Key and value are $\widehat{\mathbf{H}}''$ comes from the output of the Transformer at level-3 of the patch decoder. Noted that $\mathbf{H}'$ as the query because the output shape of the cross-attention matches the shape of the query and the length of $\mathbf{H}'$ is exactly what we want to restore from $\widehat{\mathbf{H}}''$. Specifically, $\mathbf{H}'$ only contributes to the computation of the attention matrix. The actual restored trajectory embedding is computed by weighted addition with $\widehat{\mathbf{H}}''$, ensuring that $\mathbf{H}'$ is solely to assist reconstruction.

As the sequence length of restored $\widehat{\mathbf{E}}'$ increases, we apply a self-attention to enable the restored trajectory representation to capture global information over the extended length as follows:

$$\overline{\mathbf{E}}' = \text{SelfAttn}(\text{query} = \widehat{\mathbf{E}}', \text{key} = \widehat{\mathbf{E}}', \text{value} = \widehat{\mathbf{E}}'), \tag{9}$$

We once again utilize $\mathbf{H}'$ from the patch encoder as the shortcut connection [18], adding it with the restored trajectory embedding $\overline{\mathbf{E}}'$ as input to the Transformer, making more precise trajectory reconstruction at level-2:

$$\widehat{\mathbf{H}}' = \text{TransformerEncoder}(\overline{\mathbf{E}}' + \mathbf{H}'), \tag{10}$$

We repeat Equations(8)-(10) to obtain trajectory embedding $\widehat{\mathbf{H}} \in \mathbb{R}^{|T| \times d}$ at level-1 using $\widehat{\mathbf{H}}'$ of the patch decoder, and $\mathbf{H}$ of the patch encoder. Here, position encoding is discarded in the decoder because $\mathbf{H}'$ and $\mathbf{H}$ carry position information in the patch encoder.

4.4 Reconstruction Loss for Training

We train our model by using spatial reconstruction loss and temporal reconstruction loss with the MSE loss function. Specifically, for $\widehat{\mathbf{h}}_i \in \widehat{\mathbf{H}}$, we first use two MLP networks to compute spatial and temporal predictions as follows:

$$y_i^s = \text{MLP}_1(\widehat{\mathbf{h}}_i) \in \mathbb{R}^6, \quad y_i^t = \text{MLP}_2(\widehat{\mathbf{h}}_i) \in \mathbb{R}^6, \tag{11}$$

then we compute spatial-temporal loss for trajectory T as follows:

$$\mathcal{L}^T = \sum_{i=1}^{|T|} (y_i^s - s_i)^2 + \sum_{i=1}^{|T|} (y_i^t - t_i)^2. \tag{12}$$

We average the loss over the $\mathcal{N}$ trajectories in a mini-batch to obtain the final reconstruction loss $\mathcal{L}$.

Unlike previous studies that only focus on spatial loss [20, 26, 31], we incorporate a temporal loss to enhance the reconstruction accuracy of low-level trajectories. Moreover, MSE loss is more efficient compared to the masked language model (MLM) loss [9] used in the grid- and road-based methods [20, 31], as MLM relies on classification with cross-entropy. Given that road segments and grids typically number in the tens of thousands, the Softmax operation in cross-entropy becomes highly inefficient [33].

Due to page limit, we analyze the complexity of BLUE in Appendix A.2, and the results show that the overall complexity of BLUE is $O(L_1 n_1^2)$, where L_1 and n_1 are the Transformer layers and sequence length of level-1, respectively. In Section 5.3, we show by experiment that BLUE is competitive in efficiency when compared with SOTA TRL methods.

Table 1: Statistics of the experiment datasets.

Dataset	#Traj.	Avg. length	Area size	Start time	End time
Porto	1,133,593	4,158meters	78.3km^2	2013/07/01	2014/06/30
Chengdu	1,114,194	4,702meters	708.6km^2	2014/08/03	2014/08/05

5 Experimental Evaluation

We experiment extensively to answer the following questions:

- **RQ1**: How does BLUE's accuracy compare with state-of-the-art TRL methods for various downstream tasks?
- **RQ2**: How do BLUE's designs, i.e., hierarchical patches, and different pooling methods contribute to accuracy?
- **RQ3**: How accurate is BLUE for cross-dataset model transfer?
- **RQ4**: How efficient is BLUE for training and inference?
- **RQ5**: How do BLUE's hyperparameters affect accuracy?

5.1 Experiment Settings

Datasets. We experiment on two real-world trajectory datasets, i.e., Porto[2] and Chengdu[3], which are widely used by previous TRL studies [20, 26, 31]. The proportions of training, validation, and testing data are set to [0.6, 0.2, 0.2] for both datasets. The statistics of the datasets are shown in Table 1. GPS points include latitudes, longitudes, and timestamps, which are sampled every 15 seconds and 30 seconds in Porto and Chengdu on average, respectively. Porto records 3 different travel modes for vehicles: 1) The trip is dispatched from the central; 2) The trip is demanded directly to a taxi driver on a specific stand; 3) Otherwise (e.g., a trip demanded on a random street). Chengdu includes whether a taxi was carrying a

[2]https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i
[3]https://www.pkbigdata.com/common/zhzgbCmptDetails.html

Table 2: Accuracy of BLUE and the baselines for the 3 downstream tasks on the Porto dataset. The best-performing baseline is marked with an underline, and the bottom row is the relative improvement of BLUE over the best baseline.

	Travel Time Estimation (Seconds)			Most Similar Trajectory Search			Trajectory Classification	
	MAE↓	MAPE(%)↓	RMSE↓	MR↓	HR@1↑	HR@5↑	Micro-F1↑	Macro-F1↑
Traj2vec	159.533	28.349	172.453	24.677	0.366	0.556	0.686	0.650
TrajCL	64.731	11.776	89.798	3.543	0.785	0.885	0.823	0.795
Trembr	92.665	18.021	121.967	8.345	0.582	0.713	0.810	0.753
PIM	105.987	22.091	132.983	16.453	0.462	0.587	0.770	0.714
JCLRNT	98.356	20.014	129.786	11.754	0.540	0.648	0.802	0.742
START	83.355	15.158	116.624	2.932	0.801	0.854	0.819	0.772
MMTEC	62.673	11.593	87.072	2.023	0.831	0.927	0.819	0.784
JGRM	57.776	10.865	82.533	1.832	0.858	0.949	0.813	0.775
BLUE	**6.756**	**1.142**	**12.060**	**1.136**	**0.911**	**0.998**	**0.833**	**0.796**
Imp.	88.31%	89.49%	85.39%	37.99%	6.18%	5.16%	1.22%	0.13%

Table 3: Accuracy of BLUE and the baselines for the 3 downstream tasks on the Chengdu dataset. The best-performing baseline is marked with an underline, and the bottom row is the relative improvement of BLUE over the best baseline.

	Travel Time Estimation (Seconds)			Most Similar Trajectory Search			Trajectory Classification		
	MAE↓	MAPE(%)↓	RMSE↓	MR↓	HR@1↑	HR@5↑	F1↑	Accuracy↑	Precision↑
Traj2vec	162.653	32.734	187.973	33.345	0.347	0.503	0.757	0.671	0.713
TrajCL	79.383	15.614	117.153	6.287	0.664	0.766	0.872	0.802	0.828
Trembr	94.435	18.794	127.546	9.345	0.532	0.673	0.842	0.788	0.812
PIM	116.954	22.895	139.866	19.423	0.403	0.536	0.805	0.743	0.763
JCLRNT	102.323	21.088	134.768	13.532	0.492	0.636	0.824	0.763	0.791
START	85.741	16.163	119.917	3.447	0.710	0.828	0.884	0.826	0.857
MMTEC	69.983	15.012	95.642	2.406	0.762	0.907	0.849	0.786	0.814
JGRM	66.453	13.456	93.846	2.006	0.784	0.951	0.868	0.792	0.818
BLUE	**29.302**	**5.428**	**46.299**	**1.441**	**0.831**	**0.983**	**0.904**	**0.855**	**0.874**
Imp.	55.91%	59.66%	50.66%	28.17%	5.99%	3.36%	2.26%	3.51%	1.98%

passenger. After preprocessing (in Appendix A.3), we get 1,133,593 and 1,114,194 trajectories in Porto and Chengdu, respectively.

Baselines. We compare BLUE with 8 state-of-the-art TRL methods, *Traj2vec* [45], *TrajCL* [3], *Trembr* [13], *PIM* [41], *JCLRNT* [32], *START* [20], *MMTEC* [26], *JGRM* [31]. Among the baselines, TrajCL is the best-performing grid-based method, START is the best-performing road-based method, and JGRM is the best-performing multi-modal method. More details are provided in Appendix A.4.

Implementation. We implement BLUE on an Nvidia A6000 48GB GPU running Ubuntu 22.04LTS with PyTorch 2.4. The hidden embedding and trajectory representation dimensions are set to $d = 128$. The Transformer configuration includes: 2 layers at level-1, 4 layers at level-2, and 2 layers at level-3 for both the patch encoder and decoder. Each attention mechanism uses 4 heads with a dropout ratio of 0.1. We pre-train BLUE with the Adam [22] optimizer, using a batch size of 256 and a learning rate of 1e-4. Following START [20], all methods are pre-trained for 30 epochs. The maximum patch length M is dynamically computed in a batch.

Downstream Tasks and Accuracy Measures. We chose three downstream tasks that are widely used in TRL: travel time estimation [32], trajectory classification [20], and most trajectory similarity search [3]. We do not include trajectory recovery [8] or trajectory prediction [39] as downstream tasks because grid-based and road-based methods treat these as classification problems, while GPS-based methods (including ours) treat them as regression problems, thus their evaluation metrics differ, making direct comparisons infeasible. More detailed settings are in Appendix A.5.

We follow existing TRL works [20, 26, 31] to choose the same accuracy measures for these tasks. In particular, for travel time estimation, we report the mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE). For multi-class trajectory classification on the Porto dataset, we report

Table 4: Accuracy of BLUE when disabling the key designs. We denote travel time estimation as TTE, trajectory classification as TC, and most similar trajectory search as MSTS.

	Porto			Chengdu		
	MAE↓ (TTE)	MR↓ (MSTS)	Micro-F1↑ (TC)	MAE↓ (TTE)	MR↓ (MSTS)	F1↑ (TC)
w/o P@2	7.089	1.219	0.820	29.581	2.187	0.891
w/o P@3	6.932	1.389	0.809	30.564	4.619	0.901
w/o P@5	7.418	20.677	0.815	34.025	14.047	0.884
w/o Patch	10.082	1578.228	0.763	34.503	4322.761	0.878
w/ Min	7.708	1.166	0.822	35.729	2.568	0.900
w/ Max	7.054	1.161	0.822	30.735	1.878	0.902
w/ Mean	7.485	1.165	0.821	29.407	2.131	0.901
BLUE	**6.756**	**1.136**	**0.833**	**29.302**	**1.441**	**0.904**

Micro-F1 and Macro-F1. For binary trajectory classification on the Chengdu dataset, we report the F1-score, accuracy, and precision. For most similar trajectory search, we report the hit ratio at the top-k results, i.e., the HR@1, HR@5, and the mean rank (MR).

5.2 Main Results (RQ1)

Table 2 and Table 3 compare the accuracy of BLUE with the 8 baselines. We make several observations as follows.

In detail, the improvements of BLUE over the best-performing baseline are often substantial. For the travel time estimation task, the performance gain ranges from a minimum of 50.66% to a maximum of 88.31%. Notably, on the Porto dataset, our travel time estimation achieves an MAE in the single digits (measured in seconds) and a remarkably low MAPE of just 1%. The performance of the TTE task is highly correlated with the sampling rate. Unlike road-based and grid-based methods, which group multiple GPS points into a road segment or a grid cell, leading to the loss of significant spatial-temporal details, e.g., sampling information. BLUE's bottom layer preserves the full GPS spatial-temporal information, resulting in superior performance. Compared to the Porto dataset, the performance on the Chengdu dataset is slightly lower due to its highly uneven sampling rate, which ranges from 3 seconds to 60 seconds. In contrast, the sampling rate of the Porto dataset is typically around 15 seconds, providing more consistency.

For the most similar trajectory search task, BLUE achieves improvements ranging from 3.36% to over 37.99%. Although the irregular spatial-temporal nature of GPS trajectories typically hinders similarity search performance, BLUE mitigates this issue through blurred encoding and pyramid structure, transforming GPS trajectories into patch trajectories. This process reduces spatial-temporal irregularities, while the high-level trajectory representations effectively compress trajectory information and leverage low-level fine-grained details to capture hierarchical regional patterns, resulting in significantly improved similarity search performance.

For the trajectory classification task, BLUE's improvement is less pronounced compared to other tasks, as classification is relatively easier and primarily depends on overall trajectory semantics (e.g., OD and travel patterns), which are less reliant on fine-grained details. Additionally, the Chengdu dataset features binary classification, resulting in consistently high performance across all methods, as reflected by the strong baseline accuracy.

5.3 Micro Experiments

Ablation Study (RQ2). To validate BLUE's designs, we experiment with 7 of its variants:

- **w/o P@2** removes the precision@2, the final trajectory representation is the [CLS] embedding of precision@3.
- **w/o P@3** removes the precision@3. The patch trajectory of precision@2 comes from precision@5.
- **w/o P@5** removes the precision@5. We only remove the Transformer of precision@5 to keep the input as GPS trajectories.
- **w/o Patch** removes patching module. We only keep the Transformer of level-1 and spatial-temporal reconstruction task.
- **w/ Min** replaces attention-pooling with min-pooling.
- **w/ Max** replaces attention-pooling with max-pooling.
- **w/ Mean** replaces attention-pooling with mean-pooling.

Table 4 presents the results of the ablation study. For clarity, we report a single accuracy measure for each task. The results indicate that all designs contribute effectively to improving model performance. Overall, the impact on MSTS is most significant, as it directly relies on the quality of the pre-trained model. In contrast, tasks such as TTE and TC have less performance degradation due to fine-tuning. Specifically, the low-level information, i.e., P@5, has the most substantial impact on task performance, as it captures the essential spatial-temporal correlations within trajectories through the Transformer. Higher levels, on the other hand, have a relatively smaller effect on overall performance. This is because higher levels are primarily dependent on the lower-level module, and the trajectory's semantic information is heavily compressed at these levels. Removing the patch module has little impact on TTE performance because GPS trajectories still retain the sampling rate information. However, it significantly degrades similarity performance, as GPS trajectories have low travel semantic density and are inherently irregular. In addition, different pooling strategies also influence the model performance. Min and Max pooling methods focus on extracting the most dominant features of the trajectory's spatial-temporal information. Mean pooling retains more of the trajectory's spatial-temporal integrity but can lead to feature smoothing, which may result in a loss of spatial-temporal details.

Transferability Study (RQ3). We conducted experiments to evaluate the transferability of BLUE compared to SOTA methods. The results, presented in Table 5, reveal significant performance degradation in all baseline models. This is primarily because road-based and grid-based methods rely on re-initializing grid IDs and road IDs when the road network scale or grid scale differs across cities, leading to reduced performance. Notably, for MSTS tasks that do not require fine-tuning, this re-initialization introduces a pronounced performance penalty. For BLUE, there is minimal performance loss across all tasks. This is because BLUE relies solely on unified normalized GPS and time information as input, without any external dependencies, granting the model strong versatility. Notably, BLUE can transfer effectively to tasks like MSTS that do not require fine-tuning. However, when transferring from Porto to Chengdu, there is a noticeable loss in MR due to Porto's smaller city scale compared to Chengdu, resulting in data distributions that fail to cover the full range of Chengdu's trajectories. Conversely, the transfer performance from Chengdu to Porto in MSTS surpasses the results

Table 5: Accuracy for cross-dataset transfer, where the model is trained on one dataset and evaluated on another. For each method, the first row shows the accuracy on the target dataset, and the second row shows the relative change w.r.t. the source dataset. Red indicates an improvement over the no-transfer setting, while blue denotes a performance drop. highlights the best relative change in transferability.

	Chengdu→Porto			Porto→Chengdu		
	MAE↓ (TTE)	MR↓ (MSTS)	Micro-F1↑ (TC)	MAE↓ (TTE)	MR↓ (MSTS)	F1↑ (TC)
TrajCL	70.288	179.949	0.815	86.997	201.164	0.854
	8.58%	4979%	0.97%	9.59%	3100%	2.06%
START	95.234	262.764	0.793	94.583	302.430	0.866
	14.25%	8862%	3.17%	10.31%	8674%	2.04%
MMTEC	74.213	102.433	0.808	82.533	113.324	0.838
	18.41%	4963%	4.83%	17.93%	4610%	1.30%
JGRM	70.493	91.453	0.802	86.212	120.065	0.824
	22.01%	4892%	1.35%	29.73%	5885%	5.07%
BLUE	**7.556**	**1.063**	**0.825**	**30.262**	**10.503**	**0.902**
	11.84%	6.43%	0.96%	3.28%	629%	0.22%

Table 6: Performance comparison of transferability on the Roma dataset with two SOTA methods.

Method	Transfer Setting	MAE↓(TTE)	MR↓(MSTS)
TrajCL	No transfer	129.33	8.18
	Chengdu→Roma	158.37	240.26
	Porto→Roma	160.11	399.85
JGRM	No transfer	85.00	4.94
	Chengdu→Roma	111.09	201.74
	Porto→Roma	104.06	253.21
BLUE	No transfer	**57.22**	**3.14**
	Chengdu→Roma	**52.58**	**1.11**
	Porto→Roma	**56.39**	**1.98**

without transfer, suggesting that models trained on larger cities may yield superior results when transferred to smaller cities.

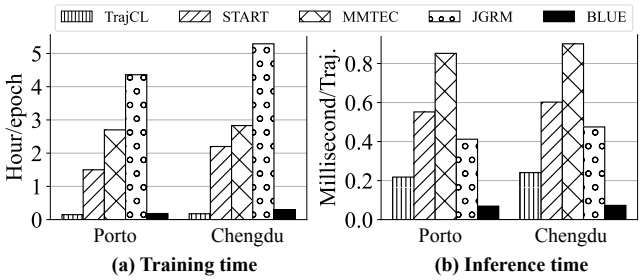
To further explore the transferability of our model, we introduce a new dataset (Roma[4] with 90,172 trajectories) to test BLUE. Since Roma lacks classification labels, we evaluate the TTE and MSTS tasks. The dataset is split into train/val/test as [0.8, 0.1, 0.1], with 1,000 samples for the query/label set and the other 50,000 samples for the database set on the MSTS task. The results of Table 6 show that BLUE outperforms two SOTA baselines in both non-transfer and transfer settings. We also can observe that the accuracy of model transfer (from other cities to Roma) can be higher than the model trained on Roma. This is because the Chengdu and Porto datasets are much larger than the Roma dataset and thus allow for pre-training higher-quality models.

Efficiency Study (RQ4). We evaluate the efficiency of our model using three metrics: model size, training time, and inference time.

[4]https://ieee-dataport.org/open-access/crawdad-romataxi

Table 7: The number of model parameters (M: million).

	TrajCL	START	MMTEC	JGRM	**BLUE**
Porto	8.18M	8.10M	1.65M	10.13M	**3.51M**
Chengdu	28.16M	15.54M	5.36M	28.73M	



(a) Training time **(b) Inference time**

Figure 5: Training time (in hours) for one epoch and model inference time (in ms) of one trajectory on batch size 256.

The results, presented in Table 7 and Figure 5, demonstrate the high efficiency of BLUE. Since our model has no external dependencies, its size remains unaffected by the scale of the city. In contrast, the parameters of road-based and grid-based models increase substantially as city size grows. Despite incorporating 20 attention layers, our model achieves fast training times and efficient inference, largely due to its pyramid structure, which drastically reduces sequence lengths at higher levels. Notably, inference is exceptionally fast as it requires only the encoder. Additionally, the MSE loss in our model is more computationally efficient than the MLM loss with Softmax operations employed by START and JGRM. While TrajCL benefits from its lightweight, i.e., two-layer networks. MMTEC requires longer training times due to its reliance on NeuralCDE and its computationally complex network structure.

Hyper-parameter Study (RQ5). We also explore how the hyper-parameters, including embedding dimensions and network layers, affect the performance of BLUE. Due to the page limit, we report the results in Appendix A.6.

6 Conclusion

We propose BLUE, a self-supervised method for trajectory representation learning through varying GPS decimal precisions. BLUE is an encoder-decoder model with a pyramid structure. The encoder compresses GPS trajectories into patch trajectories from low-level to high-level, while the decoder restores GPS trajectories from high-level to low-level. Based on the pyramid structure, BLUE can learn hierarchical semantics of patch trajectories with multiple levels. Experimental results demonstrate that BLUE consistently outperforms existing TRL methods in both effectiveness and efficiency.

7 Acknowledgement

This paper was supported by the National Key R&D Program of China 2024YFE0111800, and NSFC U22B2037, U21B2046, and 62032001.

References

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). In *arXiv preprint*. http://arxiv.org/abs/1803.08375

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. In *arXiv preprint*. http://arxiv.org/abs/1607.06450.

[3] Yanchuan Chang, Jianzhong Qi, Yuxuan Liang, and Egemen Tanin. 2023. Contrastive Trajectory Similarity Learning with Dual-Feature Attention. In *ICDE*. 2933–2945.

[4] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. 2020. A Survey on Map-Matching Algorithms. In *ADC*, Vol. 12008. 121–133.

[5] Wei Chen, Yuxuan Liang, Yuanshao Zhu, Yanchuan Chang, Kang Luo, Haomin Wen, Lei Li, Yanwei Yu, Qingsong Wen, Chao Chen, Kai Zheng, Yunjun Gao, Xiaofang Zhou, and Yu Zheng. 2024. Deep Learning for Trajectory Data Management and Mining: A Survey and Beyond. In *arXiv preprint*. https://arxiv.org/abs/2403.14151.

[6] Yile Chen, Weiming Huang, Kaiqi Zhao, Yue Jiang, and Gao Cong. 2024. Self-supervised Learning for Geospatial AI: A Survey. In *arXiv preprint*. https://doi.org/10.48550/arXiv.2408.12133.

[7] Yile Chen, Xiucheng Li, Gao Cong, Zhifeng Bao, Cheng Long, Yiding Liu, Arun Kumar Chandran, and Richard Ellison. 2021. Robust Road Network Representation Learning: When Traffic Patterns Meet Traveling Semantics. In *CIKM*. 211–220.

[8] Yuqi Chen, Hanyuan Zhang, Weiwei Sun, and Baihua Zheng. 2023. RNTrajRec: Road Network Enhanced Trajectory Recovery with Spatial-Temporal Transformer. In *ICDE*. 829–842.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*. 4171–4186.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*. https://openreview.net/forum?id=YicbFdNTTy.

[11] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. In *arXiv preprint*. http://arxiv.org/abs/1603.07285.

[12] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. 2021. E^2DTC: An End to End Deep Trajectory Clustering Framework via Self-Training. In *ICDE*. 696–707.

[13] Tao-Yang Fu and Wang-Chien Lee. 2020. Trembr: Exploring road networks for trajectory representation learning. *ACM Transactions on Intelligent Systems and Technology* 11, 1 (2020), 1–25.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.

[15] Jie Gui, Tuo Chen, Jing Zhang, Qiong Cao, Zhenan Sun, Hao Luo, and Dacheng Tao. 2024. A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (2024), 9052–9071.

[16] Jindong Han, Hao Liu, Shui Liu, Xi Chen, Naiqiang Tan, Hua Chai, and Hui Xiong. 2023. iETA: A robust and scalable incremental learning framework for time-of-arrival estimation. In *KDD*. 4100–4111.

[17] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A graph-based approach for trajectory similarity computation in spatial networks. In *KDD*. 556–564.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[19] Danlei Hu, Lu Chen, Hanxi Fang, Ziquan Fang, Tianyi Li, and Yunjun Gao. 2024. Spatio-Temporal Trajectory Similarity Measures: A Comprehensive Survey and Quantitative Study. *IEEE Transactions on Knowledge and Data Engineering* 36, 5 (2024), 2191–2212.

[20] Jiawei Jiang, Dayan Pan, Houxing Ren, Xiaohan Jiang, Chao Li, and Jingyuan Wang. 2023. Self-supervised Trajectory Representation Learning with Temporal Regularities and Travel Semantics. In *ICDE*. 843–855.

[21] Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. 2020. Neural Controlled Differential Equations for Irregular Time Series. In *NeurIPS*.

[22] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*. https://arxiv.org/abs/1412.6980.

[23] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. 617–628.

[24] Yi Li, Weiming Huang, Gao Cong, Hao Wang, and Zheng Wang. 2023. Urban Region Representation Learning with OpenStreetMap Building Footprints. In *KDD*. 1363–1373.

[25] Yuxuan Liang, Kun Ouyang, Yiwei Wang, Xu Liu, Hongyang Chen, Junbo Zhang, Yu Zheng, and Roger Zimmermann. 2022. TrajFormer: Efficient Trajectory Classification with Transformers. In *CIKM*. 1229–1237.

[26] Yan Lin, Huaiyu Wan, Shengnan Guo, Jilin Hu, Christian S. Jensen, and Youfang Lin. 2024. Pre-Training General Trajectory Embeddings With Maximum Multi-View Entropy Coding. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 9037–9050.

[27] Yan Lin, Zeyu Zhou, Yicheng Liu, Haochen Lv, Haomin Wen, Tianyi Li, Yushuai Li, Christian S. Jensen, Shengnan Guo, Youfang Lin, and Huaiyu Wan. 2024. UniTE: A Survey and Unified Pipeline for Pre-Training Spatiotemporal Trajectory Embeddings. *IEEE Transactions on Knowledge and Data Engineering* 37, 3 (2024), 1475–1494.

[28] Hao Liu, Wenzhao Jiang, Shui Liu, and Xi Chen. 2023. Uncertainty-Aware Probabilistic Travel Time Prediction for On-Demand Ride-Hailing at DiDi. In *KDD*. 4516–4526.

[29] Xiang Liu, Xiaoying Tan, Yuchun Guo, Yishuai Chen, and Zhe Zhang. 2022. CSTRM: Contrastive Self-Supervised Trajectory Representation Model for trajectory similarity computation. *Computer Communications* 185 (2022), 159–167.

[30] Yu Liu, Qian Ge, Wei Luo, Qiang Huang, Lei Zou, Haixu Wang, Xin Li, and Chang Liu. 2024. GraphMM: Graph-Based Vehicular Map Matching by Leveraging Trajectory and Road Correlations. *IEEE Transactions on Knowledge and Data Engineering* 36, 1 (2024), 184–198.

[31] Zhipeng Ma, Zheyan Tu, Xinhai Chen, Yan Zhang, Deguo Xia, Guyue Zhou, Yilun Chen, Yu Zheng, and Jiangtao Gong. 2024. More Than Routing: Joint GPS and Route Modeling for Refine Trajectory Representation Learning. In *WWW*. 3064–3075.

[32] Zhenyu Mao, Ziyue Li, Dedong Li, Lei Bai, and Rui Zhao. 2022. Jointly contrastive representation learning on road network and trajectory. In *CIKM*. 1501–1510.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

[34] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *ICLR*. https://openreview.net/forum?id=Jbdc0vTOcol.

[35] Yueyang Su, Di Yao, Xiaolei Zhou, Yuxuan Zhang, Yunxia Fan, Lu Bai, and Jingping Bi. 2023. TripSafe: Retrieving Safety-related Abnormal Trips in Real-time with Trajectory Data. In *SIGIR*. 2446–2450.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *NIPS*. 3104–3112.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.

[38] Xingrui Wang, Xinyu Liu, Ziteng Lu, and Hanfang Yang. 2021. Large scale GPS trajectory generation using map based on two stage GAN. *Journal of Data Science* 19, 1 (2021), 126–141.

[39] Hao Xue, Flora D. Salim, Yongli Ren, and Nuria Oliver. 2021. MobTCast: Leveraging Auxiliary Trajectory Forecasting for Human Mobility Prediction. In *NeurIPS*. 30380–30391.

[40] Can Yang and Gyözö Gidófalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *International Journal of Geographical Information Science* 32, 3 (2018), 547–570.

[41] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Jian Tang, and Bin Yang. 2021. Unsupervised Path Representation Learning with Curriculum Negative Sampling. In *IJCAI*. 3286–3292.

[42] Sean Bin Yang, Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S Jensen. 2023. Lightpath: Lightweight and scalable path representation learning. In *KDD*. 2999–3010.

[43] Di Yao, Haonan Hu, Lun Du, Gao Cong, Shi Han, and Jingping Bi. 2022. TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation. In *KDD*. 2275–2285.

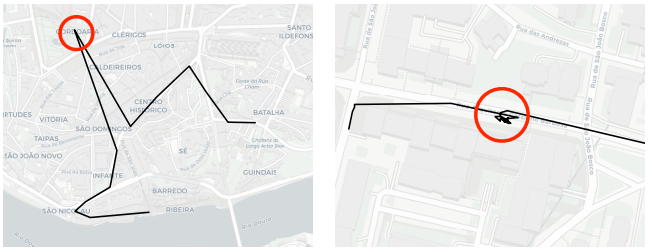
[44] Di Yao, Jin Wang, Wenjie Chen, Fangda Guo, Peng Han, and Jingping Bi. 2024. Deep Dirichlet Process Mixture Model for Non-parametric Trajectory Clustering. In *ICDE*. 4449–4462.

[45] Di Yao, Chao Zhang, Zhihua Zhu, Jian-Hui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *IJCNN*. 3880–3887.

[46] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *ECCV*. 818–833.

[47] Silin Zhou, Jing Li, Hao Wang, Shuo Shang, and Peng Han. 2023. GRLSTM: Trajectory Similarity Computation with Graph-Based Residual LSTM. In *AAAI*. 4972–4980.

[48] Zhiqiang Zou, Zhe Yu, and Kai Cao. 2017. An innovative GPS trajectory data based model for geographic recommendation service. *Trans. GIS* 21, 5 (2017), 880–896.



(a) Drift point (b) Redundant points

Figure 6: The illustration of drift and redundant points.

Table 8: Average length of different types of trajectories.

Trajectory Category	Porto	Chengdu
Road traj. (#Road segments)	27	28
Grid traj. (#Grid cells)	45	32
Level-1 traj. (#GPS points)	39	47
Level-2 traj. (#Patches)	27	31
Level-3 traj. (#Patches)	6	6

A Appendix

A.1 Algorithm

We provide an open-source pytorch-based implementation of blurred encoding. Specifically, this implementation avoids the for-loop operation in the patch encoder caused by the sequence that records all actual patch lengths of trajectories in a batch.

A.2 Complexity Analysis

The time complexity of BLUE is mainly derived from the attention mechanism of the Transformer, which is given by $O(n^2)$, where n is the sequence length. Suppose the patch trajectory lengths at level-1, level-2, and level-3 are n_1, n_2, and n_3, respectively, with corresponding Transformer layers L_1, L_2, and L_3, where $n_3 \ll n_2 \ll n_1$. The training time complexity of BLUE, including the patch encoder $O(L_1 \cdot n_1^2 + L_2 \cdot n_2^2 + L_3 \cdot n_3^2)$ and the patch decoder $O(L_1 \cdot n_1^2 + L_2 \cdot n_2^2 + L_3 \cdot n_3^2 + n_1 \cdot n_2 + n_2 \cdot n_3 + n_2^2 + n_3^2)$, where $n_1 \cdot n_2$ and $n_2 \cdot n_3$ are cross-attention, n_2^2 and n_3^2 are self-attention. The inference time complexity of BLUE, only including the patch encoder, is $O(L_1 \cdot n_1^2 + L_2 \cdot n_2^2 + L_3 \cdot n_3^2)$. Due to the reduction in trajectory length, the actual operation becomes highly efficient, i.e., approximates to $O(L_1 \cdot n_1^2)$. Table 8 in Appendix presents the average sequence lengths of different types of trajectories. The average sequence length of the patch trajectories decreases significantly, with the average length at level-3 being a single digit.

A.3 Data Preprocessing

Like previous studies [20, 31], we preprocess datasets as follows: 1) we remove trajectories less than 1 kilometer in length. 2) For the Chengdu dataset, we sample a sub-dataset. We also remove patch trajectories at precision@2 whose sequence lengths are less than 2 to ensure that patch trajectories at precision@2 are valid sequences. Moreover, considering the noise of the GPS trajectory, we also do the following processing: 1) Delete drift points. Drift points, as shown in Figure 6(a), cause significant interference with the travel semantics, and also make the wrong road trajectories and grid trajectories for baselines. We remove the drift points directly based on the abnormal driving speed[5]. 2) Redundant points, as illustrated in Figure 6(b), refer to dozens or even hundreds of GPS points clustered within a very small area, resulting in significant information redundancy and unnecessarily extending the GPS trajectory sequence. This redundancy hinders effective model learning. Notably, both grid cells and road segments provide efficient solutions to this issue. Grid cells aggregate these clustered points into a grid cell, while road segments align them to a road segment, thereby reducing redundancy. To address this issue in GPS trajectories, we take a radius of 50 meters of GPS points as a small region, and if there are more than 10 points in the region, we only keep the first point and the last point[6] to reduce redundancy. The timestamps of these two points can retain information such as congested segments.

A.4 Baselines

GPS-based Methods:

- **Traj2vec** [45][7]: An RNN-based seq2seq model converts GPS trajectory into GPS feature sequence to learn trajectory representations for the trajectory clustering task.

Grid-based Methods:

- **TrajCL** [3][8]: The state-of-the-art grid-based method proposes a set of trajectory augmentations on grid trajectory in free space and dual-feature self-attention to learn grid trajectory representations using contrastive learning with the Transformer. We set grid size as 100m×100m.

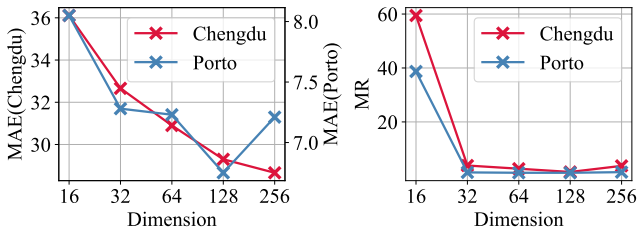
Road-based Methods:

- **Trembr** [13]: An RNN-based seq2seq model uses spatial-temporal road trajectory sequences to learn trajectory representation.
- **PIM** [41][9]: It first uses node2vec [14] to learn road embeddings on the road network and then proposes mutual information maximization to learn trajectory representations.
- **JCLRNT** [32][10]: It uses GAT and Transformer to learn road representations and trajectory representations. Three different contrastive losses are designed to optimize the model.
- **START** [20][11]: This method uses BERT [9] as the trajectory encoder that integrates travel semantics and temporal regularities with contrastive learning and two self-supervised tasks.

Multi-modal Methods:

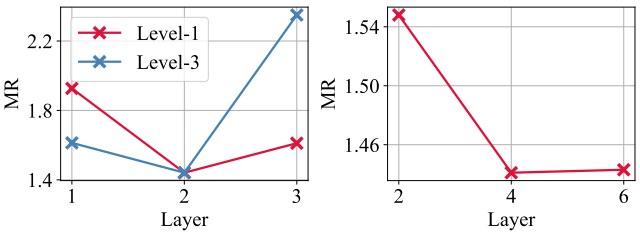
- **MMTEC** [26][12]: The multi-modal method learns trajectory representations using an attention-based discrete encoder and a NeuralCDE continuous encoder that extract travel behavior and continuous spatial-temporal correlations of trajectories.
- **JGRM** [31][13]: This multi-modal method learns trajectory representations both in the view of free-space and the view of the

[5] https://github.com/ni1o1/transbigdata
[6] https://github.com/movingpandas/movingpandas
[7] https://github.com/yaodi833/trajectory2vec
[8] https://github.com/changyanchuan/TrajCL
[9] https://github.com/Sean-Bin-Yang/Path-InfoMax
[10] https://github.com/mzy94/JCLRNT
[11] https://github.com/aptx1231/START
[12] https://github.com/Logan-Lin/MMTEC
[13] https://github.com/mamazi0131/JGRM



(a) Travel time estimation (b) Most similar search

Figure 7: Effect of embedding dimension.



(a) Level-1 and Level-3 (b) Level-2

Figure 8: Effect of layers in Chengdu on similarity.

road network space jointly with three different loss functions, i.e., contrastive loss, MLM loss, and alignment loss.

A.5 Downstream Tasks Settings

Travel Time Estimation. Travel time estimation is a fine-tuning task that aims to predict the travel time of a trajectory. This task focuses on temporal information and sampling rate information. We add a two-layer MLP after the patch encoder to predict travel time on *seconds* and use mean square error as a loss function. Since inputs of the pre-training model have full-time information, we only retain the start time and ignore other times to avoid information leakage in fine-tuning [20]. We set the learning rate to 1e-4 and other hyperparameters to be consistent with pre-training.

Trajectory Classification. Trajectory classification is also a fine-tuning task, which assigns a trajectory to a category. This task needs the pattern of movement. We classify trajectories according to the travel mode in Porto, which is a multi-classification. We determine if there are passengers in the taxi of Chengdu, which is a binary classification. We add a two-layer MLP after the patch encoder to predict the category label and use cross-entropy as a loss function. We set the learning rate to 1e-4 and other hyperparameters to be consistent with pre-training.

Most Similar Trajectory Search. Given a query trajectory T of the query dataset $\mathcal{D}_q$, the most similar trajectory search is to find the most similar trajectory T' from a large trajectory database $\mathcal{D}_d$. This task relies on spatial information (e.g., trajectory shape and origin-destination). We directly use pre-trained trajectory representations of the patch encoder with dot product of vectors to evaluate the performance (No fine-tuning). However, the lack of ground truth makes it difficult to evaluate the accuracy. Followed by TrajCL [3], we randomly select 1,000 trajectories as the query set $\mathcal{D}_q$ and another 100,000 trajectories as the database $\mathcal{D}_d$. Then, we keep the start and end GPS points unchanged to keep their overall origin-destination (OD) similar, and downsample (ratio 0.3) other GPS points in each trajectory of $\mathcal{D}_q$ to obtain sub-trajectories as the most similar trajectories set $\mathcal{D}'_q$. We add $\mathcal{D}'_q$ to $\mathcal{D}_d$ to obtain the final database set $\mathcal{D}'_d$. For baselines, we directly transform sub-trajectories to sub-grid trajectories for grid-based methods. For road-based methods, the influence of downsampling can be reduced by map matching on road trajectories [20]. Therefore, to keep the continuity of road trajectories, we adopt the detour method [32], which involves randomly selecting a middle continuous road segment constituting 30% of the length of the road trajectory and replacing it with an alternative detour.

A.6 The Effect of Hyper-parameters

Dimension. We experiment with different hidden embedding dimensions for the final trajectory representation, ranging from [16, 32, 64, 128, 256]. Figure 7 shows the results for two tasks: travel time estimation (fine-tuning) and most similar trajectory search (no fine-tuning) on two datasets. In general, both tasks improve with a larger embedding dimension, as it allows for more comprehensive encoding of travel semantics in the hidden embedding and the final trajectory representation. Specifically, when the Chengdu dataset increases from 128 to 256 dimensions, travel time estimation accuracy improves, while for the Porto dataset, accuracy decreases. This suggests that dimensional trends vary on different datasets, but overall, the 128-dimensional embedding achieves the best balance between performance and efficiency.

Layer. We vary the number of layers at each level: for level-1 and level-3, the number of layers changes between [1, 2, 3], and for level-2, it changes between [2, 4, 6]. We conducted experiments on the Chengdu dataset, and the results are shown in Figure 8. The overall trends for level-1 and level-3 are similar. When the number of layers is set to 1 or 3, performance decreases. This happens because with too few layers, the trajectory information cannot be effectively learned. For level-1, due to the limited amount of GPS trajectory information, having too many layers can lead to overfitting. For level-3, it also does not need too many layers, because the length of the patch sequence is greatly reduced. However, level-2 requires more layers since it serves as the middle layer, built from level-1 and supporting level-3. Therefore, it needs deeper modules for effective representation. Notably, the performance of level-3 stabilizes when the number of layers is set to 4.